Laravel 9 · Обработка ошибок

Введение

Когда вы запускаете новый проект Laravel, обработка ошибок и исключений уже настроена для вас. Класс `App\Exceptions\Handler` – это то место, где все исключения, созданные вашим приложением, регистрируются и затем отображаются пользователю. В этой документации мы углубимся в этот класс.

Конфигурирование

Параметр `debug` в конфигурационном файле `config/app.php` определяет, сколько информации об ошибке фактически отобразится пользователю. По умолчанию этот параметр установлен, чтобы учесть значение переменной окружения `APP_DEBUG`, которая содержится в вашем файле `.env`.

Во время локальной разработки вы должны установить для переменной окружения `APP_DEBUG` значение `true`. **Во время эксплуатации приложения это значение всегда должно быть `false`. Если в рабочем окружении будет установлено значение `true`, вы рискуете раскрыть конфиденциальные значения конфигурации конечным пользователям вашего приложения.**

Обработчик исключений

Отчет об исключениях

Все исключения обрабатываются классом `App\Exceptions\Handler`. Этот класс содержит метод `register`, в котором вы можете зарегистрировать свои отчеты об исключениях и замыкания рендеринга. Мы подробно рассмотрим каждую из этих концепций. Отчеты об исключениях используются для регистрации исключений или отправки их во внешнюю службу, например Flare, Bugsnag или Sentry. По умолчанию исключения будут регистрироваться в соответствии с вашей конфигурацией логирования. Однако вы можете регистрировать исключения как хотите.

Например, если вам нужно сообщать о различных типах исключений по-разному, вы можете использовать метод `reportable` для регистрации замыкания, которое должно быть выполнено, когда необходимо сообщить об исключении конкретного типа. Laravel определит о каком типе исключения сообщает замыкание с помощью типизации аргументов:

```
use App\Exceptions\InvalidOrderException;

/**
 * Зарегистрировать замыкания, обрабатывающие исключения приложения.
 *
 * @return void
 */
public function register()
{
    $this->reportable(function (InvalidOrderException $e) {
        //
    });
}
```

Когда вы регистрируете собственные замыкания для создания отчетов об исключениях, используя метод `reportable`, Laravel по-прежнему регистрирует исключение, используя конфигурацию логирования по умолчанию для приложения. Если вы хотите остановить распространение исключения в стек журналов по умолчанию, вы можете использовать метод `stop` при определении замыкания отчета или вернуть `false` из замыкания:

```
$this->reportable(function (InvalidOrderException $e) {
    //
})->stop();
```

```
$this->reportable(function (InvalidOrderException $e) {
    return false;
});
```

> {tip} Чтобы настроить отчет об исключениях для переданного исключения, вы можете рассмотреть возможность использования отчетных исключений.

Глобальное содержимое журнала

Если доступно, Laravel автоматически добавляет идентификатор текущего пользователя в каждое сообщение журнала исключения в качестве контекстных данных. Вы можете определить свои собственные глобальные контекстные данные, переопределив метод `context` класса `App\Exceptions\Handler` вашего приложения. Эта информация будет включена в каждое сообщение журнала исключения, написанное вашим приложением:

```
/**
 * Получить переменные контекста по умолчанию для ведения журнала.
 *
 * @return array
 */
protected function context()
{
    return array_merge(parent::context(), [
        'foo' => 'bar',
    ]);
}
```

Контекст журнала исключений

Хотя добавление контекста в каждое сообщение журнала может быть полезно, иногда конкретное исключение может иметь уникальный контекст, который вы хотели бы включить в свои журналы. Определив метод `context` для конкретного исключения вашего приложения, вы можете указать любые данные, относящиеся к этому исключению, которые должны быть добавлены в запись журнала исключения:

```
<?php

namespace App\Exceptions;

use Exception;

class InvalidOrderException extends Exception
{
    // ...

    /**
     * Получить контекстную информацию исключения.
     *
     * @return array
     */
```

```
    public function context()
    {
        return ['order_id' => $this->orderId];
    }
}
```

Помощник `report`

Иногда требуется сообщить об исключении, но продолжить обработку текущего запроса. Помощник `report` позволяет вам быстро сообщить об исключении через обработчик исключений, не отображая страницу с ошибкой для пользователя:

```
public function isValid($value)
{
    try {
        // Проверка `$value` ...
    } catch (Throwable $e) {
        report($e);

        return false;
    }
}
```

Уровни регистрации исключений

Когда сообщения записываются в журналы вашего приложения, то сообщения записываются для указанного уровня регистрации, определяющий серьезность или важность регистрируемого сообщения.

Как отмечалось выше, даже когда вы определяете пользовательское замыкание для отчета об исключении с помощью метода `reportable`, Laravel все равно будет регистрировать исключение, используя конфигурацию ведения журнала по умолчанию для приложения; поскольку уровень регистрации иногда может влиять на каналы, на которых регистрируется сообщение, то вы можете настроить уровень регистрации, на котором регистрируются определенные исключения.

Для этого вы можете определить массив типов исключений и связанных с ними уровней регистрации в свойстве `$levels` обработчика исключений вашего приложения:

```
use PDOException;
use Psr\Log\LogLevel;

/**
 * Список типов исключений с соответствующими пользовательскими уровнями регистрации.
 *
 * @var array<class-string<\Throwable>, \Psr\Log\LogLevel::*>
 */
protected $levels = [
```

```
    PDOException::class => LogLevel::CRITICAL,
];
```

Игнорирование исключений по типу

При создании приложения будут некоторые типы исключений, которые вы просто хотите игнорировать и никогда не сообщать о них. Обработчик исключений вашего приложения содержит свойство `$dontReport`, которое инициализируется пустым массивом. Ни о каких классах, добавленных в это свойство, никогда не будет сообщено; однако у них все еще может быть собственная логика отображения:

```
use App\Exceptions\InvalidOrderException;

/**
 * Список типов исключений, о которых не следует сообщать.
 *
 * @var array<int, class-string<\Throwable>>
 */
protected $dontReport = [
    InvalidOrderException::class,
];
```

> {tip} За кулисами Laravel уже игнорирует для вас некоторые типы ошибок, такие как исключения, возникающие из-за ошибок 404 HTTP «не найдено» или 419 HTTP-ответ, сгенерированный при недопустимом токене CSRF.

Отображение исключений

По умолчанию обработчик исключений Laravel будет преобразовывать исключения в HTTP-ответ за вас. Однако вы можете зарегистрировать свое замыкание для отображения исключений конкретного типа. Вы можете сделать это с помощью метода `renderable` обработчика исключений.

Замыкание, переданное методу `renderable`, должно вернуть экземпляр `Illuminate\Http\Response`, который может быть сгенерирован с помощью функции `response`. Laravel определит, какой тип исключения отображает замыкание с помощью типизации аргументов:

```
use App\Exceptions\InvalidOrderException;

/**
 * Зарегистрировать замыкания, обрабатывающие исключения приложения.
 *
 * @return void
 */
public function register()
{
    $this->renderable(function (InvalidOrderException $e, $request) {
        return response()->view('errors.invalid-order', [], 500);
```

```
    });
}
```

Вы также можете использовать метод `renderable` для переопределения поведения отображения для встроенных исключений Laravel или Symfony, таких как `NotFoundHttpException`. Если замыкание, переданное методу `renderable`, не возвращает значение, будет использовано отображение исключения Laravel по умолчанию:

```
use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;

/**
 * Зарегистрировать замыкания, обрабатывающие исключения приложения.
 *
 * @return void
 */
public function register()
{
    $this->renderable(function (NotFoundHttpException $e, $request) {
        if ($request->is('api/*')) {
            return response()->json([
                'message' => 'Record not found.'
            ], 404);
        }
    });
}
```

Отчетные и отображаемые исключения

Вместо проверки типов исключений в методе `register` обработчика исключений вы можете определить методы `report` и `render` непосредственно для ваших исключений. Если эти методы существуют, то они будут автоматически вызываться фреймворком:

```
<?php

namespace App\Exceptions;

use Exception;

class InvalidOrderException extends Exception
{
    /**
     * Сообщить об исключении.
     *
     * @return bool|null
     */
    public function report()
    {
        //
    }

    /**
```

```
     * Преобразовать исключение в HTTP-ответ.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function render($request)
    {
        return response(/* ... */);
    }
}
```

Если ваше исключение расширяет исключение, которое уже имеет методы преобразования, например, встроенное исключение Laravel или Symfony, вы можете вернуть `false` из метода `render` исключения, чтобы отобразить HTTP-ответ исключения по умолчанию:

```
/**
 * Преобразовать исключение в HTTP-ответ.
 *
 * @param  \Illuminate\Http\Request  $request
```

290 lines (217 sloc) | 20.1 KB

```
public function render($request)
{
    // Определить, требуется ли для исключения пользовательское преобразование ...

    return false;
}
```

Если ваше исключение содержит пользовательскую логику отчетности, которая необходима только при выполнении определенных условий, то вам может потребоваться указать Laravel когда сообщать об исключении, используя конфигурацию обработки исключений по умолчанию. Для этого вы можете вернуть `false` из метода `report` исключения:

```
/**
 * Сообщить об исключении.
 *
 * @return bool|null
 */
public function report()
{
    // Определить, требуется ли для исключения пользовательская отчетность ...

    return false;
}
```

> {tip} Вы можете указать любые требуемые зависимости метода `report`, и они будут автоматически внедрены в метод контейнером служб Laravel.

HTTP-исключения

Некоторые исключения описывают коды HTTP-ошибок с сервера. Например, это может быть ошибка «страница не найдена» (404), «неавторизованный доступ» (401) или даже ошибка 500, сгенерированная разработчиком. Чтобы создать такой ответ из любой точки вашего приложения, вы можете использовать глобальный помощник `abort`:

```
abort(404);
```

Пользовательские страницы ошибок HTTP

Laravel позволяет легко отображать пользовательские страницы ошибок для различных кодов состояния HTTP. Например, если вы хотите настроить страницу ошибок для кодов HTTP-состояния 404, создайте шаблон `resources/views/errors/404.blade.php`. Этот шаблон будет использоваться при отрисовки для всех ошибок 404, сгенерированных вашим приложением. Шаблоны в этом каталоге должны быть названы в соответствии с кодом состояния HTTP, которому они соответствуют. Экземпляр `Symfony\Component\HttpKernel\Exception\HttpException`, вызванный функцией `abort`, будет передан в шаблон как переменная `$exception`:

```
<h2>{{ $exception->getMessage() }}</h2>
```

Вы можете опубликовать стандартные шаблоны страниц ошибок Laravel с помощью команды `vendor:publish` Artisan. После публикации шаблонов вы можете настроить их по своему вкусу:

```
php artisan vendor:publish --tag=laravel-errors
```

Резервные страницы ошибок HTTP

Вы также можете определить «резервную» страницу ошибок для каждой серии кодов состояния HTTP. Эта страница будет отображаться, если нет соответствующей страницы для текущего кода состояния HTTP. Для этого определите шаблон `4xx.blade.php` и шаблон `5xx.blade.php` в каталоге `resources/views/errors` вашего приложения.